Computer Security CS 426

Lecture 14

Software Vulnerabilities: Format String and Integer Overflow Vulnerabilities

Format string problem

```
int  func(char *user)  {
    fprintf( stdout, user);
}
```

Problem: what if   user = “%s%s%s%s%s%s%s” ??

- Most likely program will crash: DoS.
- If not, program will print memory contents. Privacy?
- Full exploit using   user = “%n”

Correct form:

```
int  func(char *user)  {
    fprintf( stdout, “%s”, user);
}
```

Format string attacks (“%n”)

- printf(“%n”, &x) will change the value of the variable x
  - in other words, the parameter value on the stack is interpreted as a pointer to an integer value, and the place pointed by the pointer is overwritten

History

- Danger discovered in June 2000.
- Examples:

– wu-ftpd 2.* :	remote root.
– Linux rpc.statd:	remote root
– IRIX telnetd:	remote root
– BSD chpass:	local root

⋮

Vulnerable functions

Any function using a format string.

Printing:

printf, fprintf, sprintf, …

vprintf, vfprintf, vsprintf, …

Logging:

syslog, err, warn

Integer Overflow

- Integer overflow: an arithmetic operation attempts to create a numeric value that is larger than can be represented within the available storage space.
- Example:

Test 1:

```
    short x = 30000;
    short y = 30000;
    printf("%d\n", x+y);
```

Test 2:

```
    short x = 30000;
    short y = 30000;
    short z = x + y;
    printf("%d\n", z);
```

Will two programs output the same?
What will they output?

C Data Types

short int	16bits	[-32,768; 32,767]
unsigned short int	16bits	[0; 65,535]
unsigned int	16bits	[0; 4,294,967,295]
Int	32bits	[-2,147,483,648; 2,147,483,647]
long int	32 bits	[-2,147,483,648; 2,147,483,647]
signed char	8bits	[-128; 127]
unsigned char	8 bits	[0; 255]

When casting occurs in C?

- When assigning to a diffreent data type
- For binary operators +, -, *, /, %, &, |, ^,
  - if either operand is an unsigned long, both are cast to an unsigned long
  - in all other cases where both operands are 32-bits or less, the arguments are both upcast to int, and the result is an int
- For unary operators
  - ~ changes type, e.g., ~((unsigned short)0) is int
  - ++ and -- does not change type

Where Does Integer Overflow Matter?

- Allocating spaces using calculation.
- Calculating indexes into arrays
- Checking whether an overflow could occur

- Direct causes:
  - Truncation; Integer casting

Integer Overflow Vulnerabilities Example (from Phrack)

```
int main(int argc, char *argv[]) {
    unsigned short s;  int i;   char buf[80];
    if (argc < 3){ return -1; }
    i = atoi(argv[1]);   s = i;
    if(s >= 80)  {  printf(“No you don't!\n"); return -1; }
    printf("s = %d\n", s);
    memcpy(buf, argv[2], i);
    buf[i] = '\0'; printf("%s\n", buf); return 0;
}
```

Integer Overflow Vulnerabilities Example

- Example:

```
const long MAX_LEN = 20K;
Char   buf[MAX_LEN];
short len = strlen(input);
if (len < MAX_LEN)  strcpy(buf, input);
```

Can a buffer overflow attack occur?

If so, how long does input needs to be?

Another Example

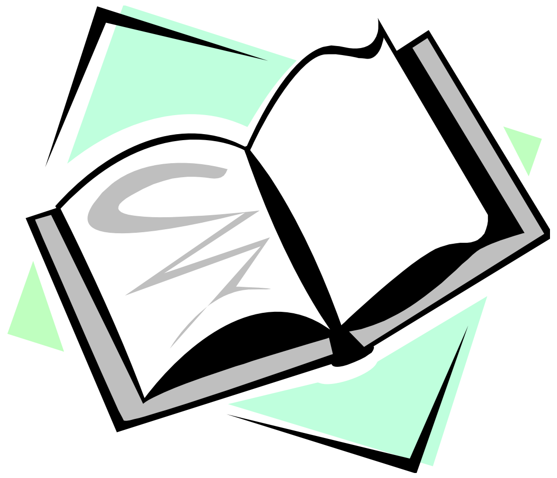
```
int ConcatBuffers(char *buf1, char *buf2,
        size_t  len1, size_t len2)
{
    char buf[0xFF];
    if ((len1 + len2) > 0xFF) return -1;
    memcpy(buf, buf1, len1);
    memcpy(buf+len1, buf2, len2);
    return 0;
}
```

Yet Another Example

```
// The function is supposed to return false when
// x+y overflows unsigned short.
// Does the function do it correctly?
bool  IsValidAddition(unsigned short x,
        unsigned short y) {
        if (x+y < x)
                return false;
        return true;
}
```

Readings for This Lecture

- Wikipedia
  - Format string attack
  - Integer overflow



Coming Attractions …

- Malwares